JEPPIAAR INSTITUTE OF TECHNOLOGY

“Self-Belief | Self Discipline | Self Respect”

DEPARTMENT

OF

COMPUTER SCIENCE AND ENGINEERING

LECTURE NOTES

CS8251 – C PROGRAMMING

(Regulation 2017)

Year/Semester: I/II CSE

2020 – 2021

Prepared by

Mr.H.Shine

Assistant Professor/CSE

Unit - 1

BASICS OF C PROGRAMMING

Introduction to programming paradigms - Structure of C Program - C programming : Data Types - Storage class - Constants - Enumeration Constants - Keywords - Operators : Precedence and Associativity - Expressions - Input/output statements, Assignment statements - Decision Making statements - Switch statement - Looping statements - Pre processor directives - Compilation process.

Introduction to programming Paradigms:-

→ Programming Paradigms are a way to Classify programming Languages based on their features. Languages can be classified into multiple paradigms.

→ Some paradigms are concerned mainly with implications for the execution model of the language, such as following side effects, or whether the sequence of operations is defined by the execution model.

⇒ Common programming Paradigms include:

* imperative which allows side effects.
* functional which disallows side effects
* declarative which does not state the order in which operations execute.

* object oriented which groups code together with the state the code modifies
* procedural which groups code into functions
* logic which has a particcular style of execut-ion model coupled to a particular style of syntax and grammar, and
* Symbolic programming which has a partic-ular style of syntax and grammer.

Machine Code:-

→ The lowest level programming paradigms are machine code, which directly represents the instructions as a sequence of numbers and assembly languages. where the machine instruct-ions are represented by mnemonics and memory address can be given Symbolic labels.

→ These are sometimes called 1st and 2nd - generation Languages.

Procedural languages:-

→ The next advance was the develope-ment of procedural languages. These 3rd generation languages uses vocabulary related to the problem ~~Solving~~ Solved. For example,

* Common Business Oriented Language (COBOL)
  → It uses terms like files, move and Copy.

* FORmula TRANslation (FORTRAN) :- Using mathematical language terminology, it was developed mainly for scientific and engineering Problems.

* ALGOrithmic Language (ALGOL) :- focused on being an appropriate language to define algorithms while using mathematical language terminology and targeting scientific and engineering Problems just Like FORTRAN.

* Programming Lanaguage One (PL/I) :- a hybrid Commercial - scientific general purpose language supporting pointers.

* Beginners All Purpose Symbolic Instruction Code (BASIC) :- It was developed to enable more people to write programs.

* C :- a general purpose programming languag initially developed by Dennis Ritchie between 1969 and 1973 at AT&T Bell Labs.

Features of C programming Language :-

✓ C is a robust language with rich set of built-in functions and operators.

✓ Programs written in C are efficient and fast.

✓ C is highly portable, programs once written in C can be run on another machines with minor or no modifications.

✓ C is basically a collection of C-library functions, we can also create our own function and add it to the C-library.

✓ C is easily extensible.

Advantages of C :-

* C is the Building block for many other programming languages.

* Programs written in C are highly portable.

* Several Standard functions are there that can be used to develop programs.

* C programs are basically Collections of C library functions, and its also easy to add own functions in to the C-library.

* The modular structure makes code debugging, maintenance and testing easier.

DisAdvantages of C:-

✓ C-doest not provide Object Oriented programming concepts (OOP).

✓ There is no concepts of Namespace in C.

✓ C-does not provide binding or wrapping up of data in a single unit.

✓ C-does not provide Constructor and Destructor.

Structure of C-Program:-

Documentation Section

Link Section

Definition Section

Global declaration Section

main() Function Section

{

Declaration Part
Executable Part

}

Subprogram Section

Function 1
Function 2
⋮
Function n

(user defined functions)

1. Documentation Section:-

✓ The documentation section Consists of a set of comment lines giving the name of the program, the author and other details, which the programmer would like to use later.

✓ /* ..... */ will be ignored by the Compiler and it has been put to add additional Comments in the program. So such lines are called Comments in the program.

```
ex:- /* This is comment
       multi line */
```

2. Link section:-

✓ The Link section provides instructions to the Compiler to the link functions from the System library Such as using #include directive.

```
ex: #include <stdio.h>
    # include <conio.h>
```

✓ It a preprocessor Command, which tells a C-Compiler to include stdio.h file before going to actual Compilation.

3. Definition Section:-

✓ The definition section defines all Symbolic Constants such using the # define directive.

```
ex: # define PI 3.14
```

✓ In above example, the PI is a Constant whole value is 3.14.

4. Global Declaration Section:-

√ There are some variables that are used in more than one function. Such variables are called global variables and are declared in the global declaration section that is outside of all the functions.

√ This section also declares all the user defined functions.

5. main ( ) Function section:-

√ Every C-program must have one main function section. This section Contains two parts;

1. Declaration Part
2. Executable Part

i) Declartion Part :- The declaration part declares all the variables used in the executable part.

ii) Executable Part :- √ There is at least one statement in the executable part. These two parts must appear between the opening and closing braces.

✓ The program execution begin at the opening brace and ends at the closing brace.

✓ The closing brace of the main functions ~~are~~ ~~generally~~ ~~placed~~ ~~immediately~~ is the logical end of the program.

✓ All Statements in the declaration

and executable part end with a semicolon.

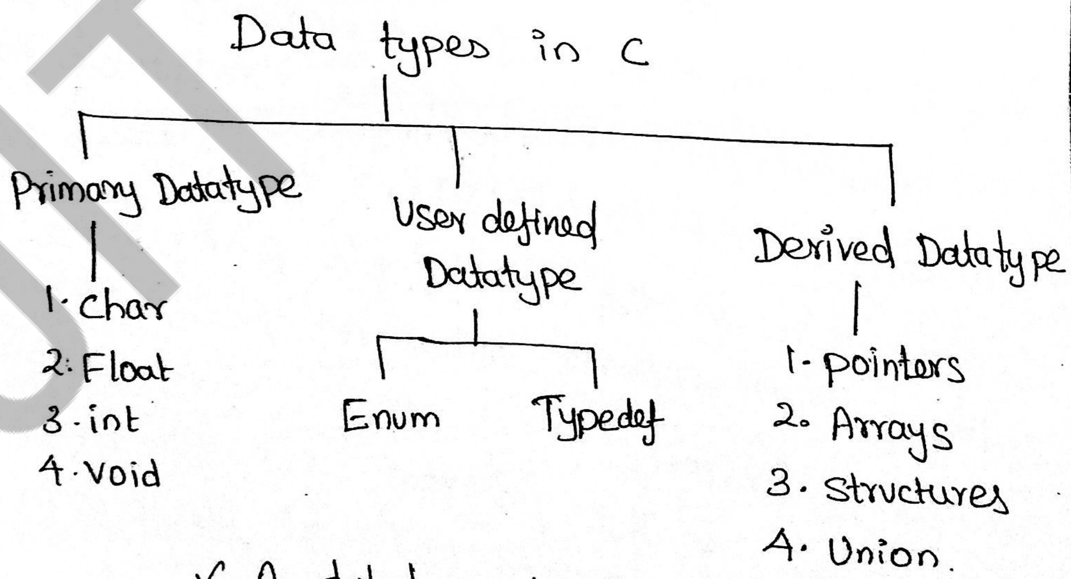
6. Subprogram Section:-

✓ If the program multi- function pro-
-gram then the subprogram section Contains all the user-defined functions that are called in the main() function.

✓ User defined functions are generally placed immediately after the main() function, althrough they may appear in any order.

C- Programming : Datatypes:-

✓ Data types specify how we enter data into our programs and what type of data we enter. These datatypes have different Storage Capacities.



✓ A datatype in C-programming is a Set of values and is determined to act on those Values.

C-datatype are used to

* Identify the type of a variable when it declared.
* Identify the type of the return value of a function
* Identify the type of a parameter expected by a function.

Declaration of Primary Data types with variable Name:-

✓ After taking suitable variable names, they need to be assigned with a data type. This is how the data types are used along with variables.

Examples:-

```
int age;
Char letter;
float height, width;
```

Data types	Memory Size	Range
Char	1 byte	-128 to 127
Signed char	1 byte	-128 to 127
Unsigned char	1 byte	0 to 255
Short	2 byte	-32,768 to 32,767
Signed short	2 byte	-32,768 to 32,767
Unsigned short	2 byte	0 to 65,535
int	2 byte	-32,768 to 32,767
Signed int	2 byte	-32,768 to 32,767
Unsigned int	2 byte	0 to 65,535
Short int	2 byte	-32,768 to 32,767
Signed short int	2 byte	-32,768 to 32,767
Unsigned Short int	2 byte	0 to 65,535

Data types	Memory size	Range
long int	4 byte	~~0 to 65,535~~ -2,147,483,648 to 2,147,483,647
signed long int	4 byte	-2147,483,648 to 2,147,483,647.
unsigned long int	4 byte	0 to 4,294,967,295.
float	4 byte	
double	8 byte	
long double	10 byte	

<u>Example for Datatypes and variable</u> declarations:-

```
#include<stdio.h>
Void main()
 {
  int a = 4000 ; // (+)ve integer data type
  float b= 5.2341;// float data type
  char c='z' ; // Char data type
  long d= 416578 ; // long (+)ve integer data type.
  short g = 130 ; // short (+)ve integer datatype.
  double i=4.12354567 // double float datatype.
  }
```

✓ The Storage representation and machine instructions differ from machine to machine. Sizeof operator Can use to get the exact size of a type or a variable on a particular platform.

Example:-

```
#include <stdio.h>
#include <limits.h>
int main()
{
printf("storage size for int is: %d\n",
                              sizeof(int));
printf("storage size of char: %d\n", sizeof(char));
return 0;
}
```

Storage classes:-

✓ Storage classes ~~used~~ used to define scope and life time of a variable. There are four storage classes in C-programming.

1. auto
2. extern
3. static
4. ~~R~~ register

i) auto:-

✓ The auto keyword is applied to all local variables automatically. It is the default storage class that is why it is known as automatic variable.

example:-

```
#include <stdio.h>
int main()
{
int a=10;
auto int b=10; // same like above
printf("%d %d",a,b);
return 0;
}
```

```
output:-
10 10
```

Storage Class	Stage Place	Default Value	Scope	Life-time
auto	RAM	Garbage Value	Local	within function.
extern	RAM	Zero	Global	Till the end of main program, May be declared anywhere in the Program.
static	RAM	Zero	Local	Till the end of main program, Retains value between multiple functions call.
register	Register	Garbage Value	Local	within function.

2) extern :-

✓ The extern variable is visible to all the programs.

✓ It is used if two or more files are sharing same variable or function.

```
ex:-
    extern int Counter=0;
```

3) Static :-

✓ The static variable is initialized only once and exists till the end of the program.

✓ It retains its value between multiple functions call.

✓ The static variable has the default value which is provided by compiler.

Example:-

```
#include <stdio.h>
int func ()
{
   static int i=0; // static variable
   int j=0; // local variable
   i++;
   j++;
   printf ("i=%d and j=%d\n", i, j);
 }
 int main()
  {
    func ();
    func ();
    func ();
    return 0
  }
```

Output:-

```
i=1 and j=1
i=2 and j=1
i=3 and j=1
```

4) register:-

✓ The register variable allocates memory in register than RAM.

✓ It's size is same of register size. It has a faster access than other variables.

✓ It is recommended to use register variable only for quick access such as in counter.

✓ we can't get the address of register variable

Example:-

```
register int counter=0;
```

Constants:-

→ A constant is a value or variable that can't be changed in the program, for example: 10, 20, 'a', 3.4, "c programming" etc.

→ There are different types of Constants in C programming.

List of Constants in C:-

Constant	Example
✓ Decimal Constant	10, 20, 450 etc.
✓ Real or floating point Constant	10.3, 20.2, 450.2 etc.
✓ Octal Constant	021, 033, 046 etc.
✓ Hexadecimal Constant	0x2a, 0x7b, 0xaa etc
✓ Character Constant	'a', 'b', 'x' etc.
✓ String Constant	"c", "c program", "c in javapoint" etc.

2 ways to define Constant in C:-

→ There are two ways to define Constant in C programming.

1. Const keyword
2. # define preprocessor

i) const - keyword:-

→ The const keyword is used to define Constant in C programming.

Example:-

```
const float Pi = 3.14;
const int c = 3;
```

→ Now the value of Pi and c variables can't be changed.

```
#include <stdio.h>
int main ()
{
  const float Pi=3.14;
  printf("The value of Pi is %f", Pi);
  return 0;
}
```

output:-

```
The value of Pi is 3.1400
```

→ If you try to change the value of Pi it will return compile time error.

```
#include <stdio.h>
int main()
{
 const float Pi = 3.14 ;
  Pi = 4.562;
 printf(" The value of Pi is %f ", Pi);
 return 0;
}
```

Output:-

```
Compile Time Error: Cannot modify a
                    Const object.
```

2) #define Preprocessor:-

✓ The #define preprocessor directive is used to define constant or micro substitution.

✓ It can use any basic data type.

Syntax:-

```
#define token value
```

Example:-

```
#include <stdio.h>
#define Pi 3.14
main()
{
 printf(" The value of Pi %f", Pi);
}
```

Output:-

```
The value of Pi 3.140000
```

Backslash character constant :-

✓ C supports some character Constants having a back slash in front of it. The list of backslash characters have a specific meaning which is known to the compiler.

✓ They are also termed as "Escape Sequence"

Example:-

\t → is used to give a tab.

\n → is used to give new line.

Constants	Meaning	Constants	Meaning
\a	Beep sound	\v	Vertical Tab
\b	back space	\'	Single Quote
\f	form feed	\"	Double Quote
\n	new line	\\	back slash
\r	Carriage return	\0	null
\t	horizontal Tab		

Enumeration Constants :-

✓ An enum is keyword, it is an user defined datatype. All properties of integer are applied on Enumeration data types. So size of the enumerator data type is 2 byte.

✓ It work like the integer.

✓ It is used for Creating an user defi--ned datatype of integer. Using enum we can create sequence of integer Constant value.

Syntax:-

$$\boxed{\text{enum tagname \{value1, value2, ... Valuen\};}}$$

✓ In above syntax enum is a keyword. It is a user defined data type.

✓ In above syntax tagname is our own variable. tagname is any variable name.

✓ value1, value2 ..... are Create set of enum values.

Example:-

$$\underset{\substack{\uparrow\\ \text{Keywork}}}{\underline{\underline{\text{enum}}}}\quad \underset{\substack{\uparrow\\ \text{User defined}\\ \text{datatype}}}{\underline{\underline{\text{week}}}}\ \{\underbrace{\text{Sun, mon, Tue, wed, thu, Fri, Sat}}_{\substack{\downarrow\\ \text{value allocated}\\ \text{for "week"}}}\};$$

✓ It is Start with 0 (Zero) by default and value is incremented by 1 for the Sequential indentifiers in the list.

✓ If Constant one value is not initialized then by default sequence will be start from zero and next to generated value should be previous Constant value one.

Example program:-

```
#include <stdio.h>
#include <conio.h>
enum abc {x, y, z};
 void main()
 {
 int a;
```

```
    a=x+y+z; // 0+1+2
    printf("Sum : %d", a);
    getch();
  }
```

Output:

```
    Sum: 3
```

Keywords:-

✓ A keyword is a reserved word. You can't use it as a variable name, constant name, etc.

✓ There are only 32 reserved words in C. language.

✓ A list of 32 keywords in C. language is given below:

auto	break	case	char	const	continue	default	do
double	else	enum	extern	float	for	goto	if
int	long	register	return	short	signed	sizeof	static
struct	switch	typedef	union	unsigned	void	volatile	while

Operators : Precedence and Associativity :-

✓ Operator is a special symbol that tells the compiler to perform specific mathema-tical or logical operation.

✓ C- has wide range of operators to perform various operations.

* Arithmatic Operators
* Relational Operators
* Logical Operators
* Bitwise operators
* Assignment operators
* Ternary or Conditional operators
* Increment and Decrement operators

i) Arithmatic Operators

✓ An Arithmatic operator performs mathema-tical operations such as addition, Substraction, multiplication, division, modulo division on numerical values.

✓ Given a table shows all the arithmatic operators supported by C-Language.

✓ Lets suppose Variable a hold 8 and b hold 3.

Operator	Meaning of operator	Example a=8 ; b=3	Result
+	addition (or) Unary plus	a+b	11
−	Subtraction (or) unary minus	a−b	5
*	Multiplication	a*b	24
/	division	a/b	2
%	remainder after division (modulo division)	a%b	0

Example program:-

```
#include <stdio.h>
int main()
{
 int a=8,b=3,c;
  c=a+b;
  printf("Addition =%d", c);
  c=a-b;
  printf("Subtraction=%d", c);
  c=a*b;
  printf(" Multiplication=%d", c);
  c=a/b;
  printf("Division=%d", c);
  c=a%b;
  printf("Modulo Division=%d",c);
  return 0;
 }
```

Output:-

```
Addition = 11
Subtraction = 5
Multiplication = 24
Division = 2
Modulo Division = 0
```

2) Relational Operators:-

✓ A relational operator checks the relation-ship between two operands.

✓ If the relation is true, it returns 1; If the relation is false, it returns 0.

✓ Relational operators are used in decis-ion making and loops.

Operator	Meaning of Operator	Example a=3 ; b=5	Result
==	Equal to	a==b → False	0
>	Greater than	a>b → False	0
<	Less than	a<b → True	1
!=	Not equal to	a!=b → True	1
>=	Greater than or equal to	a>=b → False	0
<=	Less than or equal to	a<=b → True	1

Example program:-

```
#include<stdio.h>
int main ()
 {
   int a=3, b=5;
   printf("%d ==%d is %d", a, b, a==b);
   printf("%d >%d is %d", a, b, a>b);
   printf("%d < %d is %d", a,b,a<b);
   printf("%d !=%d is %d", a, b,a!=b);
```

```
    printf("%d >= %d is %d", a, b, a>=b);
    printf("%d <= %d is %d", a, b, a<=b);
    return 0;
}
```

output:

```
3 == 5   is   0
3 > 5    is   0
3 < 5    is   1
3 != 5   is   1
3 >= 5   is   0
3 <= 5   is   1
```

3) Logical Operators:-

✓An expression containing logical operator returns either 0 or 1 depending upon whether the expression results true or false.

✓ logical operators are commonly usec in decision making in C-programming.

Operator	Meaning of operator	Example a=5; b=2, c=6	Result
&&	logical AND: True only if all operands are true.	(a>b)&&(c>b) ↳True	1
||	logical OR: True only if either one operand is true	(a>b)||(a>c) ↳True ↳False	1
!	logical NOT: True only if the operand is 0.	!(a==b) ↳False	1

Truth Table of logical operator :-

		MUL	ADD	f. Comple	
a	b	a&&b	a||b	!a	!b
0	0	0	0	1	1
0	1	0	1	1	0
1	0	0	1	0	1
1	1	1	1	0	0

Example Program:-

```
#include <stdio.h>
int main ( )
 {
 int a=5, b=2, c=6, result;
 result = (a==b) && (c > b);
 printf("(a==b)&&(c>b) is %d", result);
 result = (a>b) && (c>b);
 printf("(a>b)&&(c>b) is %d", result);
 result = (a>b) || (a>c);
 printf(" (a>b) || (a>c) is %d ", result);
 result = ! (a==b);
  printf(" ! (a==b) is %d", result);
 return 0;
 }
```

Output :-

```
(a==b)&& (c>b) is 0
```

(a>b) && (c>b) is 1

(a>b) || (a>c) is 1

!(a==b) is 1

4) Bitwise Operators:-

✓ During Computation, mathematical oper--ations like : addition, Subtraction, multiplication, division, etc are Converted to bit-level which makes processing faster and saves power.

✓ Bitwise operators are used in C-prog--ramming to perform bit-level operations.

operator	Meaning of Operator	Example a=12 ; b=25	Result
&	Bitwise AND: If two operands are 1, returns 1. If either bit an operand is 0, return 0.	12→00001100 25→00011001 00001000 a&b	8
|	Bitwise OR: It returns 1, If at least one corresponding bit of 2 operands is 1.	a|b	29
^	Bitwise XOR: It returns 1, If the corresponding bits of 2 opera--nds are opposite.	12→00001100 00011001 a^b 11 001	21
~	Bitwise Complement: It is an unary operator. It changes 1 to 0 and 0 to 1.	Let a=35 35→00100011 ~35→11011100	220
<<	Shift left: It shift all bits towards left by certain number of specified bits	Let 60<<2 60→0011 1100 ← 2 111100 00	240
>>	Right shift operator: It shifts all bits towards right by certain number of specified bits	Let 60>>2 60→00111100 → 00001111	15

Example Program:-

```
#include <stdio.h>
void main()
{
 int a=12,b=25,c;

 c=a&b;
 printf("Bitwise AND is %d", c);
 c=a|b;
 printf("Bitwise OR is %d", c);
 c=a^b;
 printf("Bitwise XOR is %d", c);

 a=35;
 c=~a;
 printf("Bitwise Complement is %d", c);
 a=60;
 b=2;
 c=a<<b;
 printf("Left shift is %d", c);
 c=a>>b;
 printf("Right Shift is %d", c);
 getch();
}
```

Output:-

```
Bitwise AND is 8
Bitwise OR is 29
Bitwise XOR is 21
Bitwise Complement is 220
Left shift is 240
Right shift is 15
```

5) Assignment operators:-

✓ An assignment operator is used for assigning a value to a variable. The most Common assignment operator is =.

Operator	Example	Same as	Example b=5
=	a=b	a=b	a=5
+=	a+=b	a=a+b	a=10
-=	a-=b	a=a-b	a=5
*=	a*=b	a=a*b	a=25
/=	a/=b	a=a/b	a=5
%=	a%=b	a=a%b	a=0

Example program:-

```
# include <stdio.h>
 int main()
 {
  int a, b=5;
  a=b;
  printf("Assignment %d", a);
 a +=b;
  printf("Addition assignment %d", a);
 a-=b;
 printf(" Subtraction assignment %d", a);
 a*=b;
  printf (" Multiplication Assignment %d", a);
 a/=b;
  printf(" Division assignment %d", a);
 a%=b;
  printf(" modulo assignment %d', a);
 return0; }
```

Output:-

```
Assignment  5
Addition Assignment  10
Subtraction Assignment  5
Multiplication Assignment  25
Division Assignment  5
Modulo Assignment  0
```

6) Ternary (or) Conditional operators:-

✓ If any operator is used on three operands or variable is known as Ternary operator.

✓ It can be represented with ?. It also called as Conditional operator.

✓ It is used to reduce the number of line codes and improve the performance of application.

Syntax:-

```
Expression 1 ? Expression 2 : Expression 3;
```

✓ In above syntax expression1 is the condition.

✓ expression2 and expression3 will be either value or variable or statement.

✓ If condition will be true expression 2 will be execute otherwise expression 3 will be executed.

Example program:-

```
#include <stdio.h>
void main()
 {
    int age = 20;
    (age>=18) ? printf("You are eligible to vote") :
                    printf("you are not eligible to vote");
    getch();
  }
```

Output :-

```
you are eligible to vote.
```

7) Increment and Decrement Operators :-

✓ Increment operators are used to increased the value of the variable by one and Decrement operators are used to decrease the value of the variable by one in C-programs.

✓ Both increment and decrement operator are used on a single operand or variable, so it is called as a unary operator.

✓ Unary operators are having higher priority than the other operators it means unary operators are executed before other operators.

✓ Increment and decrement operators are cannot apply on constant.

✓ The operators are ++, --

Types of Increment Operator :-

1. pre-Increment
2. post-Increment.

i) Pre-increment (++ variable) :-

✓ In pre-increment, 1st increment the value of variable and then used inside the expression.

Syntax:-

```
++ variable ;
```

ii) Post increment (variable ++) :-

✓ In post-increment, 1st value of variable is used in the expression and then increment the value of variable.

Syntax:-

```
variable ++;
```

Example:-

```
#include <stdio.h>
#include <conio.h>
void main()
{
int x, i;
 i=10;
 x= ++i;
 printf("Pre-increment : %d", x);
 printf(" i= %d", i);

 i=10;
 x= i++;
 printf("Post-increment : %d", x);
 printf(" i= %d", i);  }
```

output:-

```
Pre increment 10
i=10
Post increment 10
i=11
```

Types of Decrement Operator:-

1. Pre-decrement
2. Post-decrement

i)Pre decrement (-- Variable):-

✓ In pre-decrement, 1st decrement the value of variable and then used inside the expression.

Syntax:-

```
--Variable;
```

ii) Post-decrement (variable --):-

✓ In post decrement, 1st value of variable is used in the expression and then decrement the value of variable.

Syntax:-

```
Variable --;
```

Example:-

```
#include <stdio.h>
#include <conio.h>
void main()
 {
int x,i;
 i=10;
x = --i;
printf (" Pre-decrement  x = %d", x);
printf (" i = %d", i);
 i=10;
 x = i--;
 Printf (" Post-decrement  x = %d", x);
 printf (" i = %d", i);

 }
```

Output:-

```
    Pre-decrement  x = 9
    i = 9
    Post-decrement  x = 10
    i = 9
```

Precedence and Associativity:-

Precedence	Operator	Operator meaning	Associativity
1	( )	function Call	Left to Right
	[ ]	array reference	
	→	structure member access	
	.	structure member access	
2.	!	negation	Right to Left.
	~	1's Complement	
	+	Unary Plus	
	−	Unary minus	
	++	increment	
	−−	Decrement	

precedence	Operator	Operator Meaning	Associativity
2	& * sizeof	address of operator. pointer. returns size of a variable type Conversion.	Right to left.
3	* / %	multiplication Division remainder	Left to Right
4	+ -	addition subtraction	Left to Right
5	<< >>	left shift Right shift	Left to Right
6	< <= > >=	less than less than or equal to greater than greater than or equal to	Left to Right
7	== !=	equal to not equal to	Left to Right
8	&	bitwise AND	Left to Right
9	^	bitwise EXCLUSIVE OR	Left to Right
10	|	bitwise OR	Left to Right
11	&&	Logical AND	Left to Right
12	||	logical OR	Left to Right
13	?:	Conditional operator	Left to Right
14	= *=	assignment assign multiplication	Right to Left

Precedence	Operator	Operator Meaning	Associativity
14	/=	assign division	Right to Left.
	%=	assign remainder	
	+=	assign addition	
	-=	assign subtraction	
	&=	assign bitwise AND	
	^=	assign bitwise XOR	
	|=	assign bitwise OR	
	<<=	assign left shift	
	>>=	assign Right shift	
15	,	Separator	Left to Right

Example:-

Solve 17 - 8/4 * 2 + 3 - 6

Soln:- $17 - \underbrace{8/4} * 2 + 3 - 6$

$17 - \underbrace{2 * 2} + 3 - 6$

$\underbrace{17 - 4} + 3 - 6$

$\underbrace{13 + 3} - 6$

$\underbrace{16 - 6}$

10

Input/output Statements:-

✓ Majority of the programs take data as input, and then after processing the processed data is being displayed which is called information.

✓ In C programming you can use scanf()

and printf() predefined function to read and print data.

Managing Input/output :-

✓ I/O operations are useful for a program to interact with users. stdlib is the standard C library for input-output operations.

✓ while dealing with input-output operations in C, there are two important streams that play their role. These are:

* Standard Input (stdin)
* Standard Output (stdout)

✓ standard input or stdin is used for taking input from devices such as the keyboards as a data stream.

✓ standard output or stdout is used for giving output to a device such as a monitor.

✓ For using I/O functionality, programmers must include stdio header file within the program.

Reading character in C :-

✓ The easiest and simplest of all I/O operations are taking a character as input by reading that character from standard input (keyboard).

✓ getchar() function Can be used to read a single character. This function is alternate to scanf() function.

Syntax:-

```
var_name = getchar();
```

Example:-

```
#include <stdio.h>
void main()
 {
   Char title;
   title = getchar();

 }
```

✓ There is another function to do that task for files: getc which is used to accept a Character from standard input.

Syntax:-

```
int getc(FILE *stream);
```

Writing Character in C:-

✓ similar to putchar() there is another function which is used to write characters but one at a time.

Syntax:-

```
putchar(var_name);
```

Example:-

```
#include <stdio.h>
void main()
 {
   Char result = 'p';
   putchar(result);
```

```
    putchar('\n');
  }
```

✓ Similarly, there is another function putc, which is used for sending a single character to the standard output.

Syntax:-

```
int putc(int c, FILE *stream);
```

Formatted Input:-

✓ It refers to an input data which has been arranged in a specific format. This is possible in C using scanf().

✓ we have already encountered this and familiar with this function.

Syntax:-

```
scanf("Control string", arg1, arg2...argn);
```

Format Specifier:-

Format Specifier	Type of Value
%d	Integer
%f	float
%lf	Double
%c	Single character
%s	String
%u	Unsigned int
%ld	Long int
%lf	long double

Example:-

```
#include <stdio.h>
void main()
 {
   int var1 = 60 ;
   int var2 = 1234 ;
   scanf ("%2d %5d", &var1, &var2);
 }
```

✓ Input data items should have to be sep-arated by spaces, tabs or new line and the punctuation makes are not counted as separators.

Reading and writing Strings in C :-

✓ There are two popular library func-tions gets() and puts() provides to deal with strings in C.

gets() :-

✓ The char *gets (Char *str) reads a line from stdin and keeps the String pointed to by the str and is terminated when the new line is ~~used~~ read or EOF is reached.

Syntax:-

$\boxed{\text{Char *gets (Char *str);}}$

✓ where str is a pointer to an array of Characters where C strings are stored.

puts():-

✓ The function - int puts (const char *str) is used to write a string to stdout but it does not include null characters.

✓ A new line character needs to be appended to the output. The declaration is

Syntax:-

```
int puts(const char *str);
```

✓where str is the string to be written in C.

Assignment Statements:-

✓ The assignment statement has the following form:

```
variable = expression/constant/variable
```

✓ Its purpose is saving the result of the expression to the right of the assignment operator to the variable on the left.

✓ Here are some rules:-

* If the type of the expression is identical to that of the variable, the result is saved in the variable.

* Otherwise, the result is converted to the type of the variable and saved there.

→ If the type of the variable is integer while the type of the result is real, the fractional part, including the decimal point, is removed making it an integer result.

→ If the type of the variable is real while the type of the result is integer, then a decimal point is appended to the integer making it a real number.

* Once a variable receives a new value, the orginal one disappears and is no more available

Example of Assignment statements:-

```
b=c ;  // b is assigned the value of c
a=9 ;  // a is assigned the value 9
b=c+5; // b is assigned the value of exp c+5
```

* The expression on the right hand side of the assignment statement can be

✓ An arithmatic expression

✓ A relational expression

✓ A logical expression

✓ A mixed expression

For example:-

```
int a;
float b, c, avg, t;
avg = (b+c)/2; //arithmatic expression
```

```
a=b && c; /* logical expression */
a=(b+c) && (b<c); /* mixed expression*/
```

X 16M

Decision Making Statements :- (or) Control or Conditional Statements.

✓ Decision making structures requires that the programmer specifies one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true and optionally, other statements to be executed if the condition to be false.

✓ C programming languages provides the following types of deeision making statements.

1. if statement
2. if..... else statement
3. nested if statements
4. switch statement
5. if....else Ladder statements.

1) if statement :-

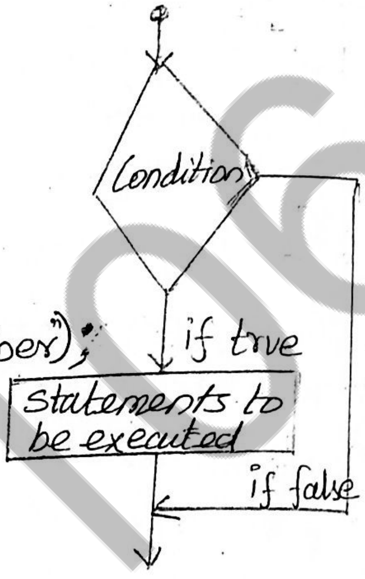
✓ The syntax of the if statement in C-progra-mming is:

```
if (test expression)
  {
  // statements to be executed if the test
                 expression is true
  }
```

✓ If the test expression is evaluated to true, statements inside the body of if are executed.

✓ If the test expression is evaluated to false, statements inside of if are not executed.

Example program:-



```
#include <stdio.h>
 int main()
  {
   int n;
   printf(" Enter the number");
   Scanf(" %d", &n);
   if(n%2==0)
    {
    printf(" The given number is even");
    }

  }
```

output 1:

```
Enter the number  12
The given number is even
The if statement is easy.
```

output 2:

```
Enter the number 15
The if statement is easy
```

2) if....else statement:-

✓ The if statement may have an optional else block. The syntax of the if... else statement is:-

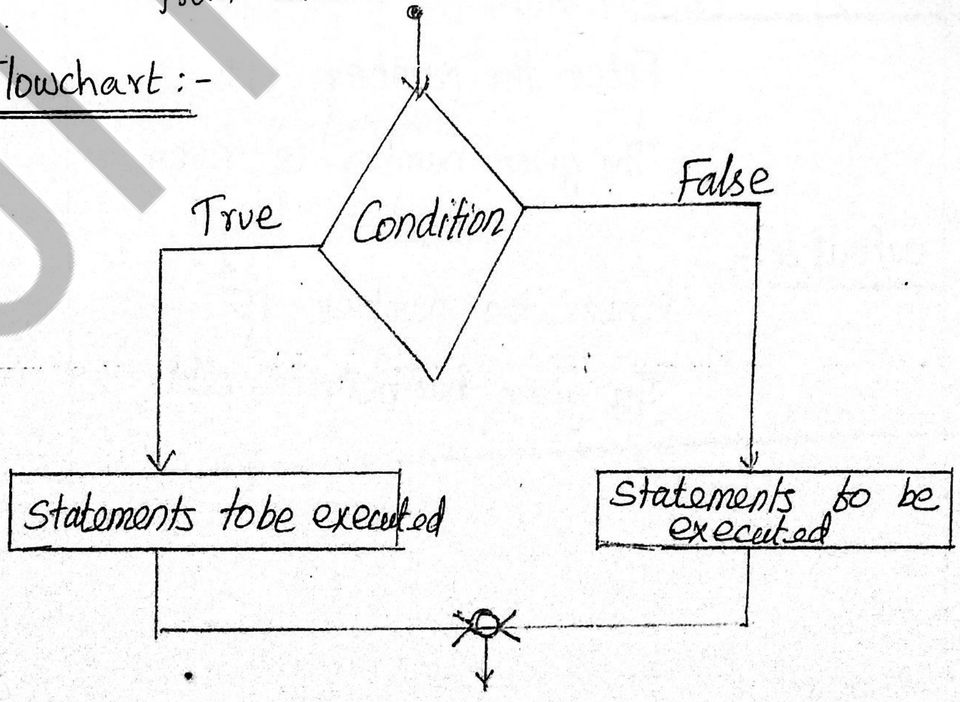
Syntax:-

```
if ( test expression)
 {
 // statements to be executed  if the test expression
              is true
 }
else
 {
 // statements to be executed if the test
             expression is false
 }
```

✓ If the test expression is evaluated to true,
 * statements inside the body of if are executed.
 * statements inside the body of else are skipped from execution.

✓ If the test expression is evaluated to false
 * statements inside the body of else are executed.
 * statements inside the body of if are skipped from execution.

Flowchart:-



Example Program:-

```
#include <stdio.h>
int main()
 {
 int : n;
 printf(" Enter the number");
 scanf("%d", &n);
  if(n%2==0)
     {
   printf(" The given number is even");
     }
  else
     {
   printf(" The given number is odd");
     }
  return 0;
 }
```

Sample output 1:-

```
Enter the number  12
The given number is even
```

output 2:-

```
Enter the number  15
The given number is odd
```

3) if.... else Ladder Statement:-

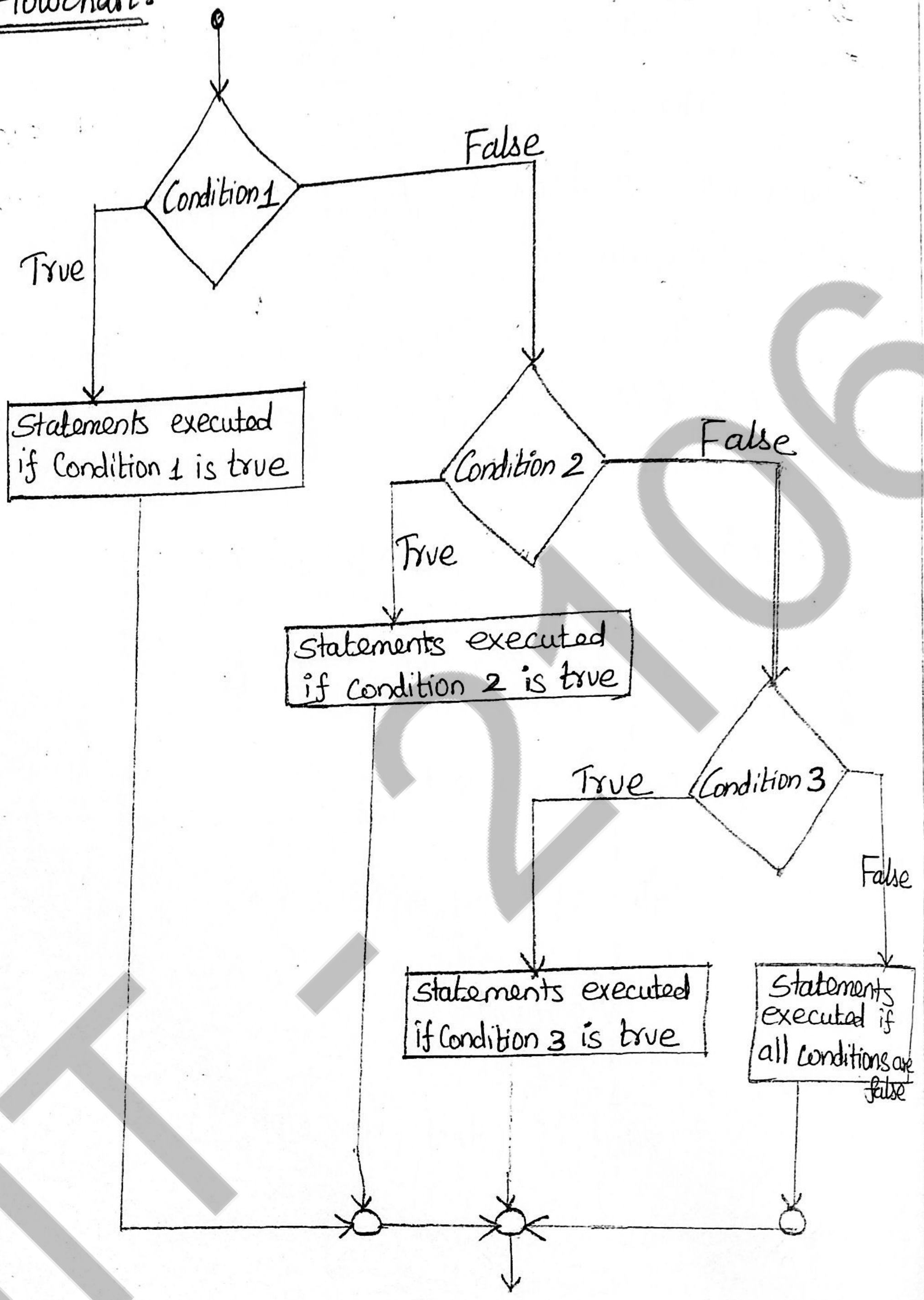
✓ The if....else ladder statement executes two different statements depending upon whether the test expression is true or false.

✓ The if... else ladder allows you to check between multiple test expression and executed different statements.

Syntax of Ladder if....else statement:-

```
if (test expression 1)
  {
   // statements ;
  }
else if (test expression 2)
  {
  // statements ;
  }
else if (test expression 3)
  {
  // statements ;
  }
else
  {
  // statements
  }
```

Flowchart:-



Condition 1
False
True
Statements executed if Condition 1 is true
Condition 2
False
True
Statements executed if condition 2 is true
Condition 3
True
False
Statements executed if Condition 3 is true
Statements executed if all conditions are false

example program1:-

```
#include <stdio.h>
int main()
{
int a,b,c;
printf("Enter the values of a,b,c");
scanf(" %d %d %d", &a, &b, &c);
 if ((a>b) && (a>c))
   {
   printf(" A is greater ");
   }
  else if (b>c)
    {
    printf (" B is greater");
    }
   else if((a==b) && (a==c))
    {
    printf (" A,b,c are equal");
    }
   else
    {
    printf (" c is greater");
    }
   return 0;
  }
```

Sample output 1 :

```
Enter the values of a,b,c   10 20 15

B is greater
```

Output 2:-

```
Enter the values of a,b,C   25 10 15

A is greater
```

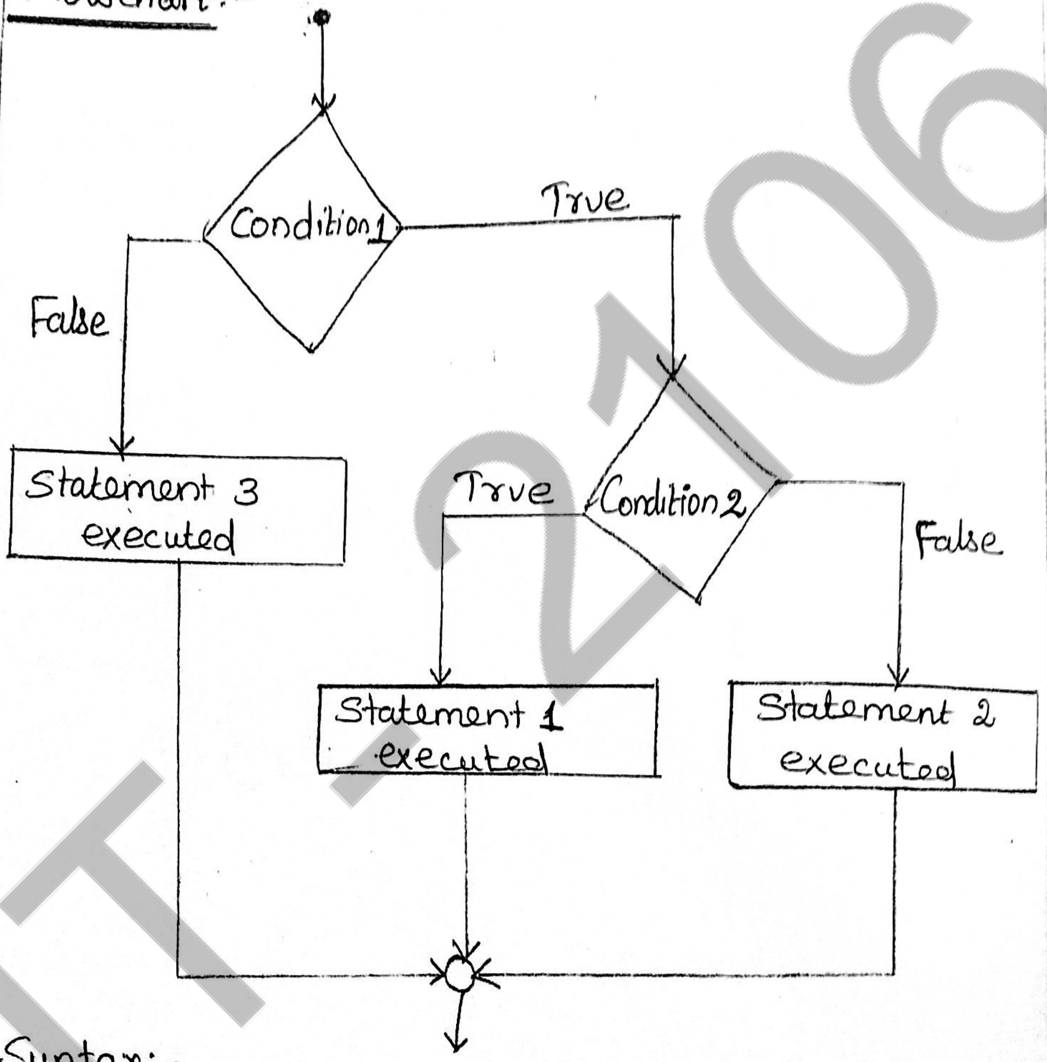
Output 3:-

```
Enter the values of a,b,C   15 10 25
C is greater
```

4) nested if...else statement:-

✓ when an if..else statement inside the body of another "if" or "else" then it is called nested if else.

Flowchart:-



Syntax:-

```
if (test expression 1)
 {
  if (test expression 2)
    {
     // Statement 1
    }
  else
    {
     Statement 2
    }
else
 {
        Statement 3
      }
```

Example program:-

```
#include<stdio.h>
int main()
{
int a, b;
printf("Enter the values of a,b");
scanf("%d %d, &a, &b);
 if (a!=b)
   {
     if (a>b)
       {
        printf(" A is greater");
       }
     else
       {
        printf("B is greater");
       }
   }
  else
   {
    printf("A and B are equal");
   }
 return 0;
}
```

Sample output:-

```
Enter the values of a,b   10 20
 B is greater
```

output 2:-

```
Enter the values of a,b   20 20
 A and B are equal
```

Output 3:-

```
Enter the values of a,b   20 10
 A is greater
```

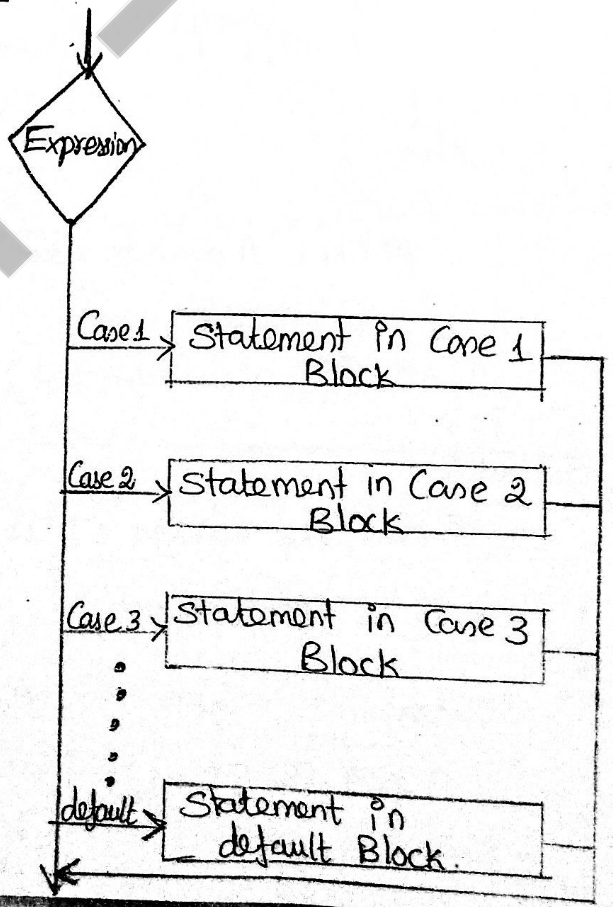
5.) Switch Statement:-

√ The switch case statement is used when we have multiple options and we need to perform a different task for each option.

Syntax:-

```
switch (variable or an integer exp)
 {
  case constant :
     // Statements
  case constant :
     // statements
  Case Constant
     // statements
  default :
     //statements
 }
```

Flowchart:-


Expression
Case 1
Statement in Case 1 Block
Case 2
Statement in Case 2 Block
Case 3
Statement in Case 3 Block
default
Statement in default Block.

Important Rules for switch case statement :-

✓ They Can have any integer Value after Case keyword. Also, case doesn't need to be inan ascending order always, you can specify them in any order as per the need of the program.

✓ you can use characters in switch case.

✓ Nesting of switch statements are allowed, Which means you can have switch statements inside another switch.

✓ Duplicate case Values are not allowed.

✓ The default Statement is optional, if you don't have a default in the program, it would exe--cute without any issues.

✓ The default statement execute if no Case is matched.

✓ Break statements are useful when you want your program-flow to come out of the switch body. Whenever a break statement is encountered in the switch body, the Control Comes out of the switch case statement.

✓ You can use any number of case Statements within a switch.

Sample program:-

```
#include <stdio.h>
int main()
{
int a,b,c,ch;
printf("1. Addition\n 2. Substraction\n
  3. Multiplication\n 4. Division\n");
printf("Enter the values of a,b");
scanf("%d %d",&a,&b);
printf("Enter your choice");
Scanf("%d",&ch);
switch (ch)
 {
 case 1:
      c=a+b;
     printf("sum=%d", c);
      break;
 Case 2:
      c=a-b;
      printf("Sub=%d", c);
      break;
 Case 3:
      c=a*b;
     Printf("MuL=%d", c);
     break;
case 4:
        c= a/b;
        printf("\n Div =%d", c);
        break;
 default:
        printf("\n Enter your Correct choice");
        break;
   }
 return 0;
}
```

Output1-

```
    1. Addition
    2. Substraction
    3. Multiplication
    4. Division
   Enter the values of a,b  20  10
   Enter your choice  1
    Sum = 30
```

Output 2:-

```
    1. Addition
    2. Substraction
    3. Multiplication
    4. Division
    Enter the values of a,b  20  10
    Enter your  choice  9
    Enter your  correct  choice
```

(X.) 13M

Looping statements :- (or) Iteration Statements

✓ Sometimes it is necessary for the program to executed the statement several times and c loops execute a block of statements a specified number of times until a condition is met.

✓ C supports following types of loop

1. for loop
2. while loop
3. do while loop.

i) for loop :-

Syntax :-

```
for(initialization ; Condition ; incr/Decr)
  {
  //statements inside the for loop
  }
```

✓ The initialization statement is executed only once

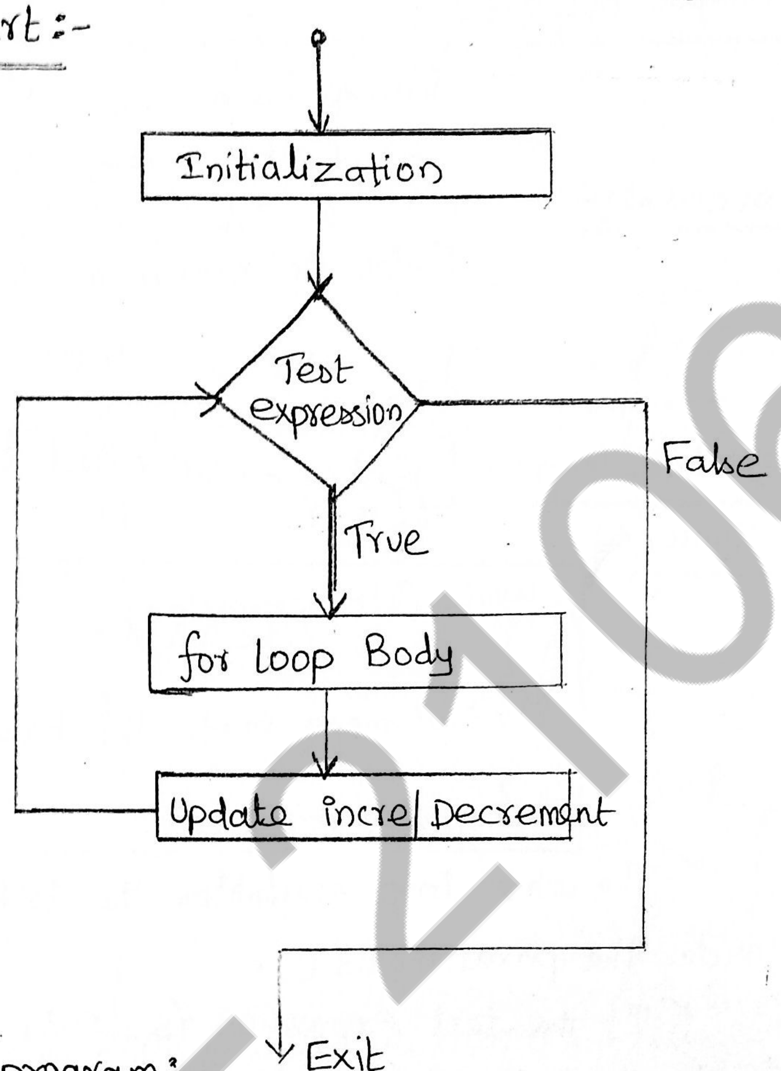
✓ Then, the condition is evaluated, if the Condition is evaluated to false, the for loop is terminated.

✓ However, if the Condition is evaluated to true, statements inside the body of for loop are executed, and the update ~~express~~ increment / Decrement statement.

✓ Again the Condition (test expression) is evaluated.

✓ This process goes on until the test expression is false. when the Condition is false, the loop terminates.

Flowchart :-



Example program :-

```
#include <stdio.h>
int main()
 {
 int i, n;
 printf("Enter the maximum limit");
 scanf("%d", &n);
  for(i=0; i<n; i++)
   {
    printf("%d", i);
   }
 return 0;
}
```

Sample output 1:-

```
Enter the maximum limit 10
0 1 2 3 4 5 6 7 8 9
```

output 2:-

```
Enter the maximum limit 6
0 1 2 3 4 5
```

2) while loop:- [Entry - Controlled loop]

Syntax:-

```
while (testexpression)
{
//Statements inside the body of whileloop
}
```

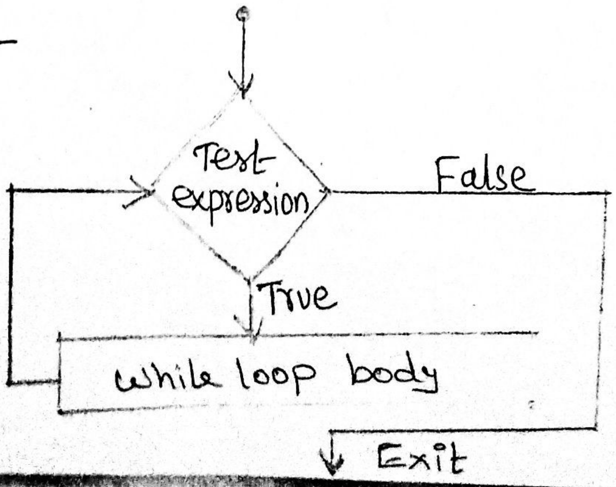
✓ The while loop evalutates the test expression inside the parenthesis ().

✓ If the test expression is true, statements inside the body of while loop are executed. Then the test expression is evaluated again.

✓ The process goes on until the test expression is evaluated to false.

✓ If the test expression is false, the loop (end) terminates.

Flowchart:-



Example program:- [Reverse Number]

```
#include<stdio.h>
int main()
 {
 int n,i,a
 printf(" Enter the number");
 scanf("%d",&n);
  Sum=0
  While (n>0)
   {
   a=n%10;
   Sum= Sum *10 + a
    n=n/10;
   }
  printf(" Reverse Number=%d", sum);
  return 0;
 }
```

Sample output 1:-

```
Enter the number    425

Reverse Number = 524
```

Output 2:-

```
Enter the number   257

Reverse Number = 752
```

3) do while loop :- [Exit-Controlled loop] :-

✓ The do....while loop is similar to the while loop with one important difference.

✓ The body of do... while loop is executed at least once. Only then, the test expression is evaluated.

Syntax :-

```
do
 {
 //statements inside the body of
                       the loop

 }
 while (test expression);
```

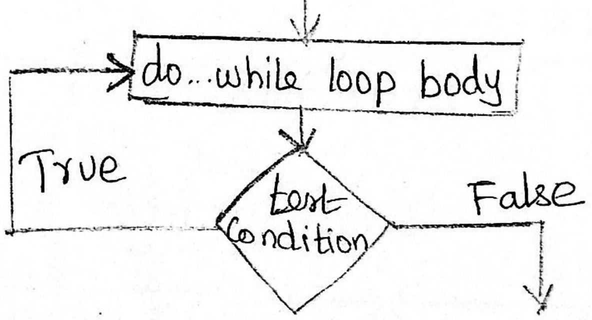
✓ The body of do...while loop is executed once. Only then, the test expression is evaluated.

✓ If the test expression is true, the body of the loop is executed again and the test expression is evaluated.

✓ This process goes on until the test expression becomes false.

✓ If the test expression is false, the loop ends.

Flowchart :-



Example program:-

```
#include <stdio.h>
int main()
{
int n, a, sum=0;
printf(" Enter the number");
scanf("%d", & n);
do
{
  a=n%10;
  sum=sum*10+a;
  n=n/10;
  } while(n>0);
printf(" The reverse number=%d", sum);
return 0;
}
```

Sample output 1:-

```
Enter the number 457
The Reverse number= 754
```

output 2:-

```
Enter the number -25
The reverse number= -5
```

C-loop Control statements:-

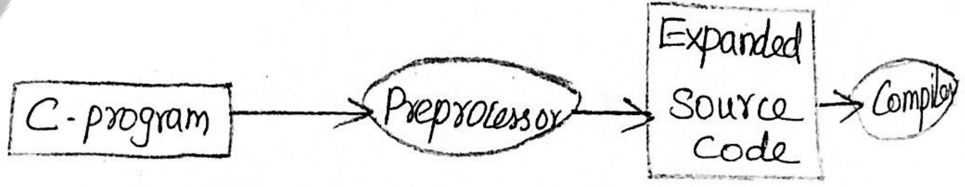
✓ loop Control statements are used to change the normal sequence of execution of the loop.

Statement	Syntax	Description
break Statement	break;	It is used to terminate loop or switch statements.
Continue Statement	continue;	It is used to suspend the execution of current loop iteration and transfer Control to the loop for the next iteration.
goto statement	goto LabelName; label Name; Statement;	It transfers current program execution sequence to some other part of the program.

Pre-Processor directives:-

✓ The C-preprocessor is a micro processor that is used by Compiler to transform your Code before Compilation. It is also micro processor because it allows us to add macros.

✓ Preprocessor directives are executed before Compilation.



✓ All preprocessor directives start with # symbol.

list of preprocessor directives

✓ #include ✓ #ifdef

✓ # define ✓ # ifndef

✓ # undef ✓ # if

✓ #else ✓ # elif

✓ #endif ✓ #error

✓ # pragma

S. NO	Preprocessor directives	Purpose	Syntax
1	#include	Used to paste code of given file into current file. It is used include System defined and user defined header files. If included file is not found, Compiler returns error.	#include<filename> #include"filename"
2	#define	used to define constant or micro substitution. It can use any basic data type.	#define PI 3.14
3.	#undef	used to undefine the constant or macro defined by #define.	#define PI 3.14 #undef PI
4.	#ifdef	checks if macro is defined by #define. If yes, it executes the code otherwise #else.	#ifdef MACRO //code #endif
5.	#ifndef	Checks if macro is not defined by #define. If yes, it executes the code otherwise #else code is executed, if present.	#ifndef MACRO //Code #endif
6.	#if	Evaluates the expression or condi-tion. If condition is true, it executes the code otherwise #elseif or #else or #endif code is executed.	#if expression //code #end if

7) #else:-

Evaluates the expression or Condition if Condition of #if is false. It can be used with #if, #elif, #ifdef and #ifndef directives.

Syntax:-

```
#if expression
    //if code
#else
    //else code
#endif
```

8) #error:-

It indicates error. The compiler gives fatal error if #error directive is found and skips.

Syntax:-

```
#error First include then compile.
```

9) # pragma:-

It is used to provide additional information to compiler. The #pragma directive is used by the compiler to offer machine or operating system feature.

Syntax:-

```
#pragma token.
```

Compilation Process:-

✓ C is a high level language and it needs a compiler to convert it into an exe-cutable code so that the program can be run on our machine.

✓ Compiler converts a C-program into an executable. There are 4 phases for a C-program to become an executable.

1. Pre-processing
2. Compilation
3. Assembly
4. Linking

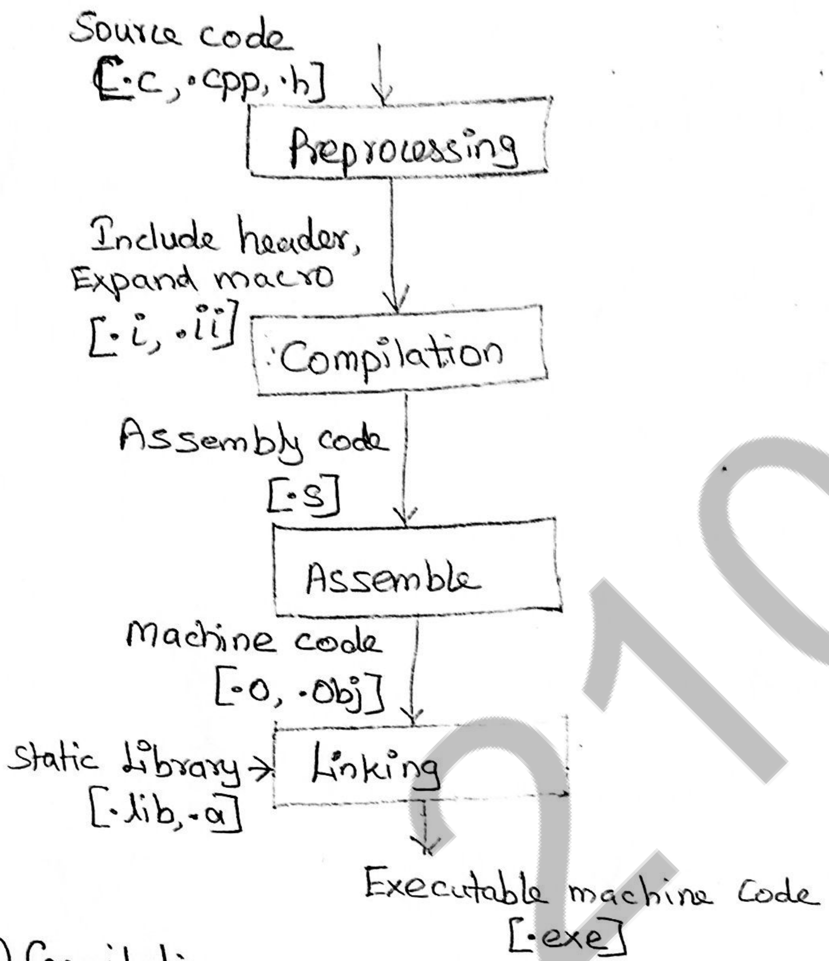
Pre-processing:-

✓ The first stage of compilation is called preprocessing.

✓ In this stage, lines starting with a # character are interpreted by the preprocessor as preprocessor commands.

✓ These commands from a simple macro language with its own syntax and semantics.

✓ This language is used to reduce repetition in source code by providing functionality to inline files, define macros, and to conditionally omit code.



(2) Compilation:-

✓ The C-compiler will convert the preprocessed source code into assembly code [·s extension].

✓ In this stage, this file is in assembly level instructions.

✓ You can see through this file using

$vi filename.s

(3) Assemble:-

✓ In this phase the filename.s is taken as input and converted into filename.o by assembler.

✓ This file contain machine level instructions.

✓ you can see or view this file using $vi filename.o.

(4) Linking:-

✓ This is final phase in which all the linking of function calls with their definitions are done.

✓ Linker knows where all functions are implementod. Linker does some extra work also, it adds some extra code to our program which is required when the program starts and ends.

✓ In the Linking step, multiple object files will be linked together to create one executable file.

✓ To produce an executable program, the existing pieces (objects) have to be re--arranged and the missing one filled in. This process is called Linking.